



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applic Serial No.:	10/645,501)	
Filing Date:	August 22, 2003)	
Applicant:	David Peyton Cox)	Appeal Brief
Examiner:	Richard Pantoliano Jr.)	
Group Art Unit:	2194)	
Title:	USING HELPER DRIVERS TO BUILD A STACK OF DEVICE OBJECTS)	
Attorney Docket No.	6215-000114)	

BRIEF ON BEHALF OF APPELLANTS

This is a brief in support of an appeal from the action of the Examiner dated December 31, 2007, finally rejecting Claims 1-28 of the present application. Copies of the appealed claims are attached as an appendix.

04/18/2008 HLE333 00000008 082025 10645501

02 FC:1402 510.00 DA

TABLE OF CONTENTS

I.	Real Parties in Interest.....	1
II.	Related Appeals and Interferences.....	2
III.	Status of Claims.....	3
IV.	Status of Amendments.....	4
V.	Summary of Claimed Subject Matter.....	5
VI.	Grounds of Rejection to be Reviewed on Appeal.....	11
VII.	Argument.....	13
VIII.	Claims Appendix.....	18
IX.	Evidence Appendix.....	27
X.	Related Proceedings Appendix.....	28

I. **Real Party In Interest**

The real party in interest in the present application is Hewlett Packard Company who is the current assignee of the application.

II. Related Appeals and Interferences

There are no known related appeals or interferences which will directly affect, be directly affected by, or otherwise have a bearing on the Board's decision in the pending appeal.

III. Status Of The Claims

Claims 1-28 are pending in the present application. Claims 1-28 stand rejected as indicated in the Advisory Action mailed on December 31, 2007. Claims 1-5 and 8-28 are the subject of this appeal; Claims 6 and 7 are not being appealed.

IV. Status Of Amendments

Applicant's response after final rejection did not propose any amendments to the pending claims. Therefore, Claims 1-28 stand as presented applicant's response dated April 12, 2007 and attached hereto in the appendix.

V. Summary of the Claimed Subject Matter

Applicant's invention is relates to a method for building a stack of device objects. Applicant's invention achieves a reduction in complexity in a different manner entirely from the WINDOWS driver model by eliminating role-discovery code from the multi-role driver's AddDevice routine. This code isn't merely moved to the several DOPush functions; it is eliminated entirely. Any particular DOPush function implies, without any question or conditions, a specific role of the multi-role driver, so when a PDO is passed to a DOPush function, the role need not be discovered.

According to Claim 1, a method is provided for building a stack of device objects (DOs) representing a device, there being a multi-role driver 402 for a plurality of roles at least one of which corresponds to the device. The method includes: registering a plurality of uni-role helper drivers 404-410 so as to uniquely correspond to the plurality of roles 438-444, respectively, where each helper driver mapping uniquely to one of the multiple roles of the multi-role driver, respectively (e.g., see paragraph [0036]); and indirectly specifying a corresponding one of the multiple roles of the multi-role driver by specifying the helper driver mapped thereto (e.g., see paragraph [0043]).

According to Claim 8, a method is provided for assembling in processor memory a stack of device objects (DOs) representing a device, there being a multi-role driver 402 for a plurality of roles at least one of which corresponds to the device, the device having a corresponding physical device object (PDO). The method includes: providing a plurality of DOPush functions 438-444 in a multi-role driver 402 (see paragraph [0034]); loading the multi-role driver into the memory so as to arrange for each of the DOPush functions

to be directly invocable by a code portion external to the multi-role driver (see paragraphs [0029] and [0035]); and invoking, externally to the multi-role driver, one of the DOPush functions, which includes passing the PDO of the device to the invoked DOPush function (e.g., see paragraph [0042]).

According to claim 15, a code arrangement is provided for assembling in processor memory a stack of device objects (DOs) representing a device (see paragraph [0023]). The machine-readable code arrangement includes: a multi-role driver code portion 402 which corresponds to the device, the multi-role driver code portion having exported functions 438-444 corresponding to the multiple roles of the multi-role driver code portion, respectively; a plurality of helper driver code portions 404-410; and an installer code portion for registering the plurality of helper driver code portions so as to uniquely map to the multiple roles, respectively (see paragraph [0037]); each helper driver code portion being operable to receive a corresponding PDO and pass the PDO to the multi-role driver code portion without attempting to attach to the stack a DO corresponding to the helper driver code portion (see paragraph [0041] and [0042]).

According to claim 20, an apparatus for building a stack of device objects (DOs) representing a device attached to the apparatus. The apparatus includes: multi-role driver means 402 for operating according to a plurality of roles (see paragraph [0029]); a plurality of helper driver means 404-410 registered so as to uniquely correspond to the plurality of roles, respectively, of the multi-role driver; and means for selectively invoking the multi-role driver according to one of the multiple roles via invoking the corresponding helper driver mapped thereto (see paragraphs [0041]-[0043]).

According to claim 23, a code arrangement is provided for building in processor memory a stack of device objects (DOs) representing a device, there being a multi-role driver 402 for a plurality of roles at least one of which corresponds to the device (see paragraph [0023]). The machine-readable code arrangement includes: a plurality of helper driver code portions 404-410; a first code portion for registering the plurality of helper driver code portions so as to uniquely correspond to the plurality of roles, respectively, each helper driver code portion mapping uniquely to one of the multiple roles of the multi-role driver, respectively (see paragraph [0037]); and a second code portion for indirectly specifying a corresponding one of the multiple roles of the multi-role driver by specifying the helper driver code portion mapped thereto (see paragraphs [0041]-[0043]).

VI. Grounds of Rejection to be Reviewed on Appeal

- I. Whether Claims 1-5 are unpatentable over applicant's admitted prior art (AAPA) in view of U.S. Patent No. 5,604,843 (Shaw) under §103(a)?
- II. Whether Claims 8-14 are unpatentable over AAPA in view of Shaw under §103(a)?
- III. Whether Claims 15-19 are unpatentable over AAPA in view of Shaw under §103(a)?
- IV. Whether Claims 20-23 are unpatentable over AAPA in view of Shaw under §103(a)?
- V. Whether Claims 24-28 are unpatentable over AAPA in view of Shaw under §103(a)?

VII. Arguments

I. Rejection of Claims 1-5 as being unpatentable over AAPA in view of Shaw.

Applicant respectfully submits that the Examiner has failed to establish a prima facie case of obviousness as required by *Graham v. John Deere Co.*, 148 USPQ 459 (1966). The key to supporting any rejection under 35 U.S.C. §103 is the clear articulation of the reasons why the claimed invention would have been obvious. The Supreme Court in *KSR International Co. v. Teleflex Inc.*, 82 USPQ2d 1385 (2007) noted that the analysis supporting a rejection should be made explicit.

Applicant's admitted prior art (AAPA) relates generally to a WINDOWS drive model that was developed by Microsoft Corporation. As conceded by the Examiner, AAPA does not disclose registering a plurality of helper drivers so as to uniquely correspond to the plurality of roles supported by a multi-role driver. The Examiner relies on Shaw to teach this aspect of Applicant's claimed invention.

Shaw relates generally to a method for interfacing with output devices, such as printers. The Examiner believes the universal driver in Shaw is analogous to the multi-role driver and the minidrivers in Shaw are analogous to helper drivers. However, these analogies are inaccurate for various reasons set forth below.

The universal driver in Shaw plays substantially the *same* role with respect to a number of instances of the same class of device, printers. The multi-role driver plays a substantially different role with respect to several distinct classes of device: SCSI busses, disk drives, and tape drives. The multi-role driver also plays multiple, substantially different roles with respect to one class of device: both lower filter and upper filter for disk

drives. Filtering below the disk FDO has an entirely different purpose than filtering above the FDO. Therefore, the Shaw universal driver does not have multiple roles in the sense that the multi-role driver does. In this regard, the Examiner's attention is drawn to claims 3, 12, 17, 22 and 26.

Shaw's universal driver exposes one set of functions, each function analogous to one listed in the graphics device interface functions list appearing in Shaw Col 2 lines 5-53. Dissimilarly, the multi-role driver exposes several sets of WDM driver functions, one set per role. Of each set, the one function primarily discussed is the "DOPush" function, similar to the AddDevice routine of a traditional WDM driver, the difference being that there can be at most one true AddDevice routine per driver, while there are in fact several DOPush functions in the multi-role driver, one per role. However, in addition to each DOPush function, there may be for each role a distinct function for handling any one of the various IO requests that may be passed to a WDM driver. For example, there may be for each role a distinct function for handling IRP_MJ_PNP requests; similarly there may be for each role a distinct function for handling IRP_MJ_WRITE requests. There are *not* similarly multiple GDI "Enable" functions or "DevBitBlt" functions in the Shaw universal driver.

In contrast, multiple functions, one per role, for handling each type of IO request, are provided for by the standard Windows Driver Model (WDM). In this model, when the OS passes a PDO to a driver's AddDevice (here DOPush) routine, the driver may decide to participate in handling IO requests for the device, or not. If it so decides, it proceeds to participate by allocating and initializing its own Device Object (DO) and attaching it to the top of (also known as pushing it onto) the stack of DOs rooted in the PDO. Each DO

contains a table of function pointers, one for each IO request type. Subsequent IO requests to the device are handled by passing them, according to their type, to one of the functions from the table of each DO in the stack, sequentially, from top to bottom. Because the multi-role driver has multiple distinct DOPush functions, one per role, it can easily initialize the table of function pointers in DOs established in each with a distinct set of request handler functions. In sum, the universal driver in Shaw is not analogous to the multi-role driver in the WINDOWS Driver Model environment. For this reason, the teachings of Shaw cannot be combined with the Window Driver Model as suggested by the Examiner.

In the Windows Driver Model, a driver's initialization routine is called before any PDOs can be passed to its AddDevice routine. The AddDevice routine *is not* the driver initialization routine. One purpose of the actual initialization routine is to populate fields in a data structure allocated by the OS called the Driver Object (not to be confused with Device Objects; there is one Driver Object loaded driver and exists independently of whether there are any devices with which the driver is associated). One such field in the Driver Object is the AddDevice function pointer. The OS cannot identify or call a driver's AddDevice routine until the driver has provided a pointer to the routine in this field. The innovation of the present application is that a helper driver may provide not a pointer to any of its own functions, but instead to a role-specific DOPush function in the multi-role driver. When the time comes for the OS to pass a PDO to a registered driver's AddDevice routine, it ends up invoking the multi-role driver's DOPush routine directly, and not involving the helper driver at all. Thus, the helper driver receives from the OS only a single invocation of its own initialization routine, and never receives any call that it must

forward to the multi-role driver. In Shaw, the minidriver always receives the calls from the OS (GDI) and forwards to the universal driver. In fact, this innovation of the present invention cannot be applied the mini/universal driver model for printer drivers. If the proposed modification or combination of the prior art would change the principle of operation of the prior art invention being modified, then the teachings of the references are not sufficient to render the claims prima facie obviousness. *In re Ratti*, 270 F.2d 810 (CCPA 1959). See also, MPEP 2143.01.VI.

As motivation to combine the relied upon references, the Examiner states that one would have been motivated by the fact that devices often share similar functions that can be implemented once and used by multiple devices ... the minidrivers taught by SHAW would allow for reduction in complexity of implementation by allowing developers to implement functions once in the universal driver and used by all of the minidrivers associated with the individual devices.” To the contrary, applicant’s invention eliminates a single AddDevice routine shared by several device types and roles as described in the AAPA in favor of several distinct DOPush functions. The multi-role driver does not have a parallel to function implemented once in the universal driver and used by all minidrivers. Besides the DOPush functions, the multi-role driver’s IO request handling routines are distinct and differ substantially in implementation from one role to another. Applicant’s invention is clearly not a case of functions implemented once and used by or for multiple similar devices.

For at least these reasons, the Examiner has failed to establish a prima facie case of obviousness. Accordingly, applicants respectfully request the Board to reconsider and withdraw this rejection.

Claim 3

Claim 3 further recites that a role is determined according to a device type for which the multi-role driver is invoked and the extent of the stack at the point at which the multi-role- driver is invoked. Shaw only deals with a single device type: printers. Each minidriver does not map uniquely to a different role of the multi-role driver. Thus, these claims help to further distinguish the helper drivers from the minidrivers disclosed in Shaw. Moreover, the unidriver does not support different functions for different device types as recited in these claims. Since the Examiner asserts that applicant's amendments necessitated the new ground of rejection, then the Examiner must be giving patentable weight to the uni-role aspect of the helper drivers. In this case, applicants respectfully request the Examiner to reconsider and withdraw these rejections. If the Examiner is not giving patentable weight to this aspect of the helper drivers, then applicant asserts the finality of the rejection is improper and should be withdrawn. For this additional reason and the reasons set forth above in relation to claim 1, claim 3 is patentable over the combination of references applied by the Examiner. Accordingly, applicants respectfully request the Board to reconsider and withdraw this rejection.

II. Rejection of Claims 8-14 as being unpatentable over AAPA in view of Shaw.

Applicant respectfully submits that the Examiner has failed to establish a prima facie case of obviousness for Claims 8-14 for at least the same reasons discussed above in relation to Claims 1-5. The Examiner is requested to consider how these arguments apply to the invention recited in Claim 8 which varies in scope from Claim 1. In other words, Claim 8 requires consideration independent from Claim 1. Accordingly, applicants respectfully

request the Board to reconsider and withdraw this rejection.

III. Rejection of Claims 15-19 as being unpatentable over AAPA in view of Shaw.

Applicant respectfully submits that the Examiner has failed to establish a prima facie case of obviousness for Claims 15-19 for at least the same reasons discussed above in relation to Claims 1-5. The Examiner is requested to consider how these arguments apply to the invention recited in Claim 15 which varies in scope from Claim 1.

In other words, Claim 15 requires consideration independent from Claim 1. In addition, Claim 15 recites that the multi-role driver code portion has exported functions corresponding to the multiple roles of the multi-role driver code portion, respectively. The Examiner does not address how the applied references teach this claim limitation. Even assuming arguendo that Shaw teaches a universal driver having multiple roles, Shaw does not export distinct functions or sets of functions for each such role. Given the number of printer models, to do so would create a prohibitive number of exported functions and tedious replication of common code. Accordingly, applicants respectfully request the Board to reconsider and withdraw this rejection.

IV. Rejection of Claims 20-23 as being unpatentable over AAPA in view of Shaw.

Applicant respectfully submits that the Examiner has failed to establish a prima facie case of obviousness for Claims 20-23 for at least the same reasons discussed above in relation to Claims 1-5. The Examiner is requested to consider how these arguments apply to the invention recited in Claim 20 which varies in scope from Claim 1.

In other words, Claim 8 requires consideration independent from Claim 1. Accordingly,

applicants respectfully request the Board to reconsider and withdraw this rejection.

V. Rejection of Claims 24-28 as being unpatentable over AAPA in view of Shaw.


Applicant respectfully submits that the Examiner has failed to establish a prima facie case of obviousness for Claims 24-28 for at least the same reasons discussed above in relation to Claims 1-5. The Examiner is requested to consider how these arguments apply to the invention recited in Claim 24 which varies in scope from Claim 1.

In other words, Claim 24 requires consideration independent from Claim 1. Accordingly, applicants respectfully request the Board to reconsider and withdraw this rejection.

For the foregoing reasons, the appealed claims are patentably distinguishable over the art relied upon by the Examiner. Accordingly, Applicant's representative respectfully requests that this Board reverse the final rejection of Claims 1-5 and 8-28.

Respectfully submitted,

Dated: April 16, 2008



Timothy D. MacIntyre
Registration No. 42,824

HARNESS, DICKEY & PIERCE
P.O.Box 828
Troy, Michigan 48303
(248) 641-1600

TDM/med

VIII. Claims Appendix

1. A method used while building in processor memory a stack of device objects (DOs) representing a device, there being a multi-role driver for a plurality of roles at least one of which corresponds to the device, the method comprising:

registering a plurality of uni-role helper drivers so as to uniquely correspond to the plurality of roles, respectively,

each helper driver mapping uniquely to one of the multiple roles of the multi-role driver, respectively; and

indirectly specifying a corresponding one of the multiple roles of the multi-role driver by specifying the helper driver mapped thereto.

2. The method of claim 1, wherein the multi-role driver and the helper drivers are operable to run in the WINDOWS Driver Model environment.

3. The method of claim 1, wherein a role is determined according to a device type for which the multi-role driver is invoked and the extent of the stack at the point at which the multi-role driver is invoked.

4. The method of claim 1, wherein each of the multiple roles in the multi-role driver has a corresponding DOPush function, each DOPush function having been made available to be invoked by a code portion external to the multi-role driver.

5. The method of claim 1, wherein each helper driver includes an AddDevice

routine that invokes a corresponding DOPush function in the multi-role driver, or each helper driver points to the address of the corresponding DOPush function in the multi-role driver.

8. A method used while assembling in processor memory a stack of device objects (DOs) representing a device, there being a multi-role driver for a plurality of roles at least one of which corresponds to the device, the device having a corresponding physical device object (PDO), the method comprising:

providing a plurality of DOPush functions in a multi-role driver;

loading the multi-role driver into the memory so as to arrange for each of the DOPush functions to be directly invocable by a code portion external to the multi-role driver; and

invoking, externally to the multi-role driver, one of the DOPush functions, which includes passing the PDO of the device to the invoked DOPush function.

9. The method of claim 8, wherein the DOPush function is invoked externally by

an AddDevice routine of a helper driver, or

the PnP manager, if the helper driver does not have the AddDevice routine, after the PnP manager is pointed to the address of the DOPush function by the helper driver,

the helper driver being registered uniquely for the role to which the DOPush function corresponds.

10. The method of claim 8, wherein the multi-role driver is operable to run in the WINDOWS Driver Model environment.

11. The method of claim 8, further comprising: registering neither the multi-role driver nor the DOPush functions in the registry of the operating system as having a role in assembly of a stack representing a device.

12. The method of claim 8, wherein a role is determined according to a device type for which the multi-role driver is invoked and the extent of the stack at the point at which the multi-role driver is invoked.

13. A method used while assembling in processor memory a stack of device objects (DOs) representing a device, the method comprising:

providing a multi-role driver for a plurality of device types; but

not registering, in the registry of the operating system, the multi-role driver as having a role in assembly of the stack.

14. The method of claim 13, wherein the multi-role driver is operable to run in the WINDOWS Driver Model environment.

15. A code arrangement on a machine-readable medium execution of which facilitates assembling in processor memory a stack of device objects (DOs) representing a device, the machine-readable code arrangement comprising:

a multi-role driver code portion which corresponds to the device, the multi-role driver code portion having exported functions corresponding to the multiple roles of the multi-role driver code portion, respectively;

a plurality of helper driver code portions; and

an installer code portion for registering the plurality of helper driver code portions so as to uniquely map to the multiple roles, respectively;

each helper driver code portion being operable to receive a corresponding PDO and pass the PDO to the multi-role driver code portion without attempting to attach to the stack a DO corresponding to the helper driver code portion.

16. The machine-readable code arrangement of claim 15, wherein the multi-driver code portion and the helper driver code portions are operable to run in the WINDOWS Driver Model environment.

17. The machine-readable code arrangement of claim 15, wherein a role is determined according to a device type for which the multi-role driver code portion is invoked and the extent of the stack at the point at which the multi-role driver code portion is invoked.

18. The machine-readable code arrangement of claim 15, wherein the exported functions are DOPush functions.

19. The machine-readable code arrangement of claim 18, wherein each helper

driver code portion includes an AddDevice routine code portion that invokes the corresponding DOPush function in the multi-role driver code portion, or each helper driver code portion is operable to point to the address of the corresponding DOPush function in the multi-role driver code portion.

20. An apparatus having memory in which is buildable a stack of device objects (DOs) representing a device attached to the apparatus, the apparatus comprising:

multi-role driver means for operating according to a plurality of roles;

a plurality of helper driver means registered so as to uniquely correspond to the plurality of roles, respectively, of the multi-role driver; and

means for selectively invoking the multi-role driver according to one of the multiple roles via invoking the corresponding helper driver mapped thereto.

21. The apparatus of claim 20, wherein the multi-role driver means and the helper driver means are operable to run in the WINDOWS Driver Model environment.

22. The apparatus of claim 20, wherein a role is determined according to a device type for which the multi-role driver means is invoked and the extent of the stack at the point at which the multi-role driver means is invoked.

23. The apparatus of claim 20, wherein the multi-role driver means provides a plurality of DOPush functions corresponding to the multiple roles, respectively, each DOPush function having been made available to be invoked by a code portion external to

the multi-role driver means.

24. A code arrangement on a machine-readable medium execution of which facilitates building in processor memory a stack of device objects (DOs) representing a device, there being a multi-role driver for a plurality of roles at least one of which corresponds to the device, the machine-readable code arrangement comprising:

a plurality of helper driver code portions;

a first code portion for registering the plurality of helper driver code portions so as to uniquely correspond to the plurality of roles, respectively, each helper driver code portion mapping uniquely to one of the multiple roles of the multi-role driver, respectively; and

a second code portion for indirectly specifying a corresponding one of the multiple roles of the multi-role driver by specifying the helper driver code portion mapped thereto.

25. The machine-readable code arrangement of claim 24, wherein the multi-role driver and the helper driver code portions are operable to run in the WINDOWS Driver Model environment.

26. The machine-readable code arrangement of claim 24, wherein a role is determined according to a device type for which the multi-role driver is invoked and the extent of the stack at the point at which the multi-role driver is invoked.

27. The machine-readable code arrangement of claim 24, wherein each of the multiple roles in the multi-role driver has a corresponding DOPush function, each DOPush function having been made available to be invoked by a code portion external to the multi-role driver.

28. The machine-readable code arrangement of claim 24, wherein each helper driver code portion includes an AddDevice routine code portion that invokes a corresponding DOPush function in the multi-role driver, or each helper driver code portion points to the address of the corresponding DOPush function in the multi-role driver.

IX. Evidence Appendix

None

X. Related Proceedings Appendix

None